Multi-Perspective Anomaly Detection in Business Process Execution Events

Kristof Böhmer and Stefanie Rinderle-Ma

University of Vienna, Faculty of Computer Science
{kristof.boehmer,stefanie.rinderle-ma}@univie.ac.at

Abstract. Ensuring anomaly-free process model executions is crucial in order to prevent fraud and security breaches. Existing anomaly detection approaches focus on the control flow, point anomalies, and struggle with false positives in the case of unexpected events. By contrast, this paper proposes an anomaly detection approach that incorporates perspectives that go beyond the control flow, such as, time and resources (i.e., to detect contextual anomalies). In addition, it is capable of dealing with unexpected process model execution events: not every unexpected event is immediately detected as anomalous, but based on a certain likelihood of occurrence, hence reducing the number of false positives. Finally, multiple events are analyzed in a combined manner in order to detect collective anomalies. The performance and applicability of the overall approach are evaluated by means of a prototypical implementation along and based on real life process execution logs from multiple domains.

Keywords: Business process events, Execution logs, Anomaly detection

1 Introduction

Protecting business processes against fraud, misuse, and unknown attacks, indicated by *anomalous process executions*, is crucial [12, 4]. Anomalous process executions can be detected by distinguishing unlikely/irregular process executions from expected executions [2]. For this, the expected executions are defined, for example, by a reference process model or based on recorded process model executions (i.e., execution events). Ongoing executions are then compared to the expected ones for identifying deviations which, in turn, might indicate anomalies.

Anomalies can be classified as follows, cf. [6]: *a*) *Point Anomalies* – a single independent event indicates an anomaly; *b*) *Contextual Anomalies* – whether an observed event is anomalous or not depends on its context, for example, when or by whom an activity is executed; *c*) *Collective Anomalies* – the combination of multiple events (e.g., resource assignments or activity execution times) enables to identify an anomaly while each independent event is inconspicuous.

Existing work mainly focuses on *point anomalies*, for example, by checking if activity execution events occur in an expected order. Moreover it was found that existing work either exclusively focuses on the *control flow* or ignores it almost completely, e.g., only analyzes activity execution times, cf. [19, 12], which could

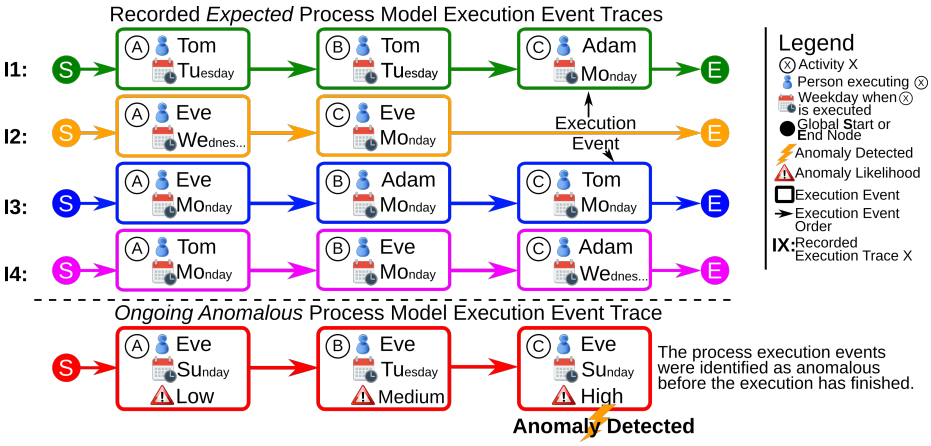

Fig. 1. Expected, and abnormal traces and events – running example.

result in undetected anomalies. A comprehensive analysis requires to examine different process perspectives at once, in order to, e.g., analyze the behavior of a user in the context of a specific activity. Considering multiple process perspectives and process execution events at once paves the way to detect *contextual anomalies* and goes beyond sheer consideration of the control flow. Consider Fig. 1. It depicts recorded process instance execution events (i.e., expected events) which are compared to events of an ongoing process execution. Events from multiple ongoing process perspectives show anomalies, for example, the organizational perspective (typically not all activities are executed by *Eve*) and the execution time perspective (never before was an activity executed on a *Sunday*).

It is also crucial to detect *collective anomalies* that occur through the combination of different events, for example, multiple events that only slightly deviate from the expected ones. When considering each event in a separated manner, small deviations might slip undetected through the anomaly detection, while their aggregated effect might result, for example, in a fraud attempt and should therefore be identified as anomalous. Hence, this paper proposes to analyze all observed events of an ongoing process execution in relation to preceding events. Consider Fig. 1 where the anomaly likelihood increases from event to event by aggregating small derivations that can be observed in the bottommost (red) process execution event trace.

On top of detecting contextual and collective anomalies, *unexpected events* often hamper the quality of the anomaly detection result. Unexpected events can result for different reasons such as incorrect process executions data (i.e., noise), non-deterministic parallel executions (e.g., parallel branches), or manual ad-hoc process model changes (e.g., a concept drift), cf. [10]. All of these reasons do not (necessarily) indicate an anomaly. However, such unexpected events are, by existing work, almost always classified as anomalous which can have a negative impact on commercial success because valid process executions could be halted or even terminated because they are incorrectly classified as anomalous.

The identified limitations motivate the following research questions:

RQ1 How to include multiple perspectives into an anomaly detection approach?

RQ2 How can contextual and collective anomalies be identified?
RQ3 How can unexpected events be handled to reduce false positives?

Here, the process execution aspects that are considered for anomaly detection are denoted as *events*. Hereby, an event can represent information from different process execution perspectives, such as, the control flow, organizational, or time perspective (e.g., an event can indicate that an specific activity a was chosen for execution or that a was executed on a specific timestamp) $\mapsto$ **RQ1**. The basic idea is to record (i.e., log) all executions, and related execution events, of a process model M. Then, the execution events of an ongoing process execution of M are compared with the recorded execution events from previous executions – to detect deviations (i.e., anomalies). Modern process execution engines enable to access the events of ongoing process executions, cf. [11], so that an ongoing execution can even be identified as anomalous before it has finished. In addition each newly observed event can be analyzed in combination with all its predecessors $\mapsto$ **RQ2**. This also enables the handling of unexpected events. Hence, each unexpected event is compared with the expected events to assess if it represents, e.g., a valid manual process adaption or a harmful security breach $\mapsto$ **RQ3**. The overall approach is evaluated by means of a prototypical implementation along and based on real life process execution logs from multiple domains.

This paper is organized as follows. The proposed approaches, to identify and prevent process execution anomalies, are discussed in Section 2. Evaluation, corresponding results and their discussion are presented in Section 3. Section 4 discusses related work. Conclusions and future work are given in Section 5.

2 Anomaly Detection in Process Execution Events

The proposed anomaly detection approach extracts the *expected execution events* for a process model M from a log L that holds M's recorded execution traces. Those traces are typically, generated automatically by today's process model execution engines [17]. L is a list of execution traces $t \in L$ where $t :=< e_1, \ldots, e_i >$ with events $e_i := (a, r, t)$. Here, a denotes the activity that e_i is associated with (i.e., control flow process perspective), r denotes the back-end system/user that has executed the activity (i.e., organizational perspective), and t represents the timestamp when the recorded event occurred (i.e., when a's execution started – time perspective). Let further e_i^+ identify the direct successor of each execution event in a trace and $\{\cdots\}^0$ return the only element stored in a set iff the set is a *singleton*. This log definition enables to support *different modeling languages* and is supported by process log formats, such as, XES or MXML, cf. [17].

A two-pronged anomaly detection approach is proposed, cf. Fig. 2. First, a representation of M's expected execution events, their occurrence and likelihoods are created. Therefore, *expected events* are collected based on recorded process execution logs ①. Note, execution logs are utilized to ensure that the presented approach represents the real expected executions events, including manual adaptions, of a process model and not, for example, some abstracted and probability outdated documentation. ② the logs are utilized to construct a *basic likelihood*

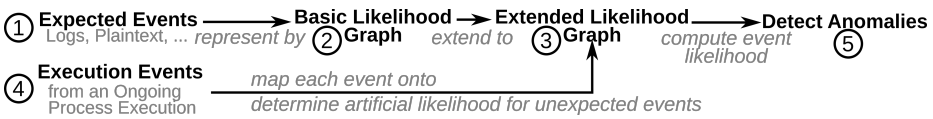

Fig. 2. Proposed anomaly detection approach – overview.

graph. The basic graph encodes the control flow, i.e., all recorded process model activities and the likelihood that one activity is followed by another activity. ③ Depending on the available perspectives (e.g., the organizational or time perspective) the basic likelihood graph is *extended* to represent the likelihoods that, for example, an activity was executed by a specific resource on a chosen weekday.

Secondly, the likelihood graph is utilized to assess if an ongoing execution event is anomalous or not. Hence each event of an *ongoing execution* ④ is *mapped* on the likelihood graph. Note, that the likelihood graph contains the likelihood of all the expected events (i.e., based on recorded execution logs) and can, therefore, be utilized to determine the likelihood of ongoing process execution events. However, novel events can occur which were never observed before and that are, therefore, not represented in the graph/logs. In such a case an *artificial likelihood* is calculated for such unexpected events. Finally, ⑤ the likelihood of the ongoing execution events and comparable events that are stored in the logs are utilized to compute if the mapped ongoing execution events are anomalous or not.

Note, the combination of the basic and the extended likelihood graph (i.e., ② and ③) enables to *a*) create an approach that can frequently be applied (because the required control flow perspective is provided by most process execution log formats [17]); and *b*) that can be, by design, extended by integrating additional perspectives that are represented by process execution events (e.g., this work will extend the basic graph based on the organizational and time perspective).

2.1 Basic Likelihood Graph

To identify anomalies (i.e., unlikely events) it is necessary to be aware of the likelihood of each event (e.g., the likelihood of a transition from one activity to another one) that can be observed during a process model execution [16]. Therefore we propose to construct a basic likelihood graph that represents the likelihood of activity execution transitions which are observable at recorded expected executions (i.e., based on recorded execution logs). Hence, the basic graph represents the control flow activity transitions of the analyzed execution logs. Note, in Sect. 2.2 it is shown how the basic graph can be extended to support/ represent additional process perspectives (e.g., resources and execution times).

Let $G = (V, D)$ denote a cyclic graph that consists of a set of vertexes $v \in V$ and a set of edges $d \in D$. Each vertex v represents an activity that occurs in the events/execution traces stored in the execution log L. Each edge $d = (v_s, v_e, l)$ represents a transition between two vertexes v_s and v_e and a related transition likelihood of l. Hence, $D \subseteq V \times V \times [0; 1]$ is a set of edges labeled with a likelihood $l \in [0; 1]$. Because the graph represents the likelihood of expected execution events (e.g., activity execution transitions) it is called "likelihood graph".

Alg. 1 creates a basic likelihood graph G from a log L by iterating through all recorded traces $t \in L$ and t's execution events $e \in t$. For each event e a new vertex is added to V (i.e., holding e's activity a) and connected to the previously processed activity a_{lst} through a likelihood edge $d(a_{lst}, a, \texttt{likeA}(L, a_{lst}, a))$. Note, the first/last graph vertex is always set to an artificial starting/ending vertex S/E. The likelihood for an identified activity transition, e.g., from a_{lst} to a is calculated by Alg. 2 (i.e., $\texttt{likeA}(\dots)$). For this it divides ec, the explicit amount of times activity a follows on activity a_{lst}, by tc, the total amount of times activity a_{lst} is executed. Note, we expect, for Alg. 1, that each element in V/D is unique, for example, if multiple traces $t \in L$ contain the same activity a, and therefore a is added multiple times to V, then a still only occurs once in V.

Activity transition likelihoods are also represented by existing work, e.g., Heuristic Nets (HN) [18]. However HN cannot be utilized here because they tend to filter out rare events to simplify the generated nets (i.e., HN do not represent all the events stored in the execution logs). Moreover, HN utilize an abstract likelihood calculation approach which results in a likelihood representation that slightly *deviates* from the "real" likelihoods. These deviations stack up during the aggregation of multiple event likelihoods which negatively affect the anomaly detection performance. Hence, we propose the presented likelihood graph.

Algorithm basicLikelihoodGraph(*execution logs* L)
- **Result:** a basic likelihood graph $G = (V, D)$
- V:=$\{S, E\}$, D:=$\emptyset$ // S,E are artificial Start/End vertexes
- **foreach** $t \in L$ **do**
  - a_{lst}:=S
  - **foreach** $e \in t$ **do**
    - V:=$V \cup \{e.a\}$
    - D:=$D \cup \{a_{lst}, e.a, \texttt{likeA}(L, a_{lst}, e.a))\}$ // likeA cf. Alg. 2
    - a_{lst}:=$e.a$
  - D:=$D \cup \{a_{lst}, E, 1)\}$ // assume a dummy likelihood of 1 for the end vertex
- **return** $G = (V, D)$

Algorithm 1: Construction of the basic likelihood graph.

Algorithm likeA(*execution logs* L, *activities* $a_s, a_e \in V$)
- **Result:** likelihood that a_e is executed after a_s
- **if** $a_s = S$ // assume a dummy likelihood of 1 for the start vertex **then**
  - **return** 1
- $tc := 0$, $ec := 0$
- **foreach** $t \in L$ **do**
  - tc:=$tc + |\{x | x \in t.e \wedge x.a = a_s\}|$
  - ec:=$ec + |\{x | x \in t.e \wedge x.a = a_s \wedge x^+.a = a_e\}|$
- **return** ec/tc

Algorithm 2: Calculation of likelihood for activities.

When applying the presented basic graph construction approach onto the running example (cf. Fig. 1) then the likelihood graph depicted in Fig. 3 is created. Note, the edge colors indicate which process execution traces (i.e., I1

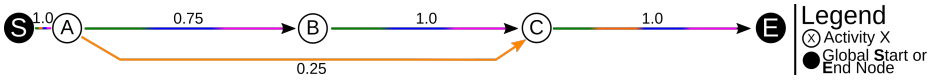

Fig. 3. Basic likelihood graph, based on the running example in Fig. 1.

to I4) are represented by an edge. Fig. 3, e.g., enables to determine activity transition likelihoods, for example, that activity A is followed by activity B with a 75% likelihood because 3 out of 4 expected traces show this activity transition.

2.2 Extending the Basic Likelihood Graph

We propose to exploit not only the control flow (reflected by the basic likelihood graph) for anomaly detection but also *additional perspectives* that are represented by process execution events and which can indicate an anomaly. Imagine, for example, an activity that was always executed during Monday or Wednesday. However suddenly the activity is executed on Sunday when "normally" all employees take a day off from work. Obviously, this kind of behavior is rather unlikely and could be anomalous. However, existing anomaly detection approaches cannot draw the same conclusion if they only concentrate on the control flow.

Hence, the basic likelihood graph is extended by *integrating additional process perspectives*, such as, the organizational, data flow, or time perspective – depending on their availability in the log L. To integrate a perspective, into the likelihood graph, its recorded execution events are converted into a countable set of *classes*. Numeric values can be, for example, represented by a set of number ranges, while timestamps can be, for example, represented as weekdays, quarters, or months. Note, that existing work can be utilized to identify diverse or unique perspectives, cf. [20], and to automatically classify the recorded perspective events to integrate them in the proposed approach, cf. [7]. This is even possible for textual data, for example, by applying regular expressions [5].

This work considers two perspectives: *a*) the organizational perspective, specifically, the resource (i.e., a user or a back-end system) that has executed an activity; and *b*) the time perspective, represented as the day of the week when an activity execution was started. These perspectives were chosen for multiple reasons. First, we found that they are not supported by existing process anomaly detection work – which could result in anomalies gone unnoticed. Secondly, these two perspectives can be observed in a variety of values which enables to show the flexibility of the presented approach. Moreover, it can be expected that these perspectives enable to detect a wide range of harmful anomalies. For example, if an attacker has stolen valid user credentials from a user u and starts to execute processes and activities then the attacker will likely execute activities that are typically not executed by u (i.e., because the attacker is not aware of u's expected behavior) or use unusual execution times (e.g., on weekends so that no security manager is actively monitoring the process executions). Note, the required perspectives are supported by today's process execution log formats, cf. [17], which this paper exploits to determine the expected execution events.

Formally, the basic graph $G := (V, D)$ is extended with a bag of perspective event class vertexes F (e.g., weekdays) so that $V := V \cup F$. Auxiliary functions,

such as, `isRes`(v) and `isWeekday`(v), determine whether v is a resource/weekday (i.e., $isRes : V \rightarrow \{T, F\}$ and $isWeekday : V \rightarrow \{T, F\}$).

Alg. 3 extends the basic likelihood graph by integrating resources and execution times. Each edge that originally only connects two activity vertexes is replaced with an series of edges and vertexes that represent the likelihood that an activity v is executed by a specific resource (i.e., back-end system or user) r, on a specific weekday wd, and subsequently followed up by an activity v_{next}. The likelihoods of each combination (e.g., that an activity is executed by a specific resource) is determined by Alg. 4 (i.e., `likeG`$(\dots)$) using the recorded execution traces $t \in L$ (i.e., the recorded process execution events). Additional perspectives can be integrated into the likelihood graph by extending Alg. 3 and Alg. 4 analogous to the existing loops/conditions.

The order of the represented perspectives was chosen based on the occurrence of events in real life process executions. Hence, we assume that first the activity that should be executed is chosen, secondly an appropriate resource must be identified, and, thirdly, the resource will start the activity on a specific weekday. Auxiliary function $getWkdy(t)$ determines the weekday of a timestamp t (i.e., $getWkdy : t \rightarrow \{Monday, Tuesday, \cdots, Sunday\}$).

Algorithm `extendLikelihoodGraph(`*a basic likelihood graph* $G = (V, D)$*, execution logs* L`)`
- **Result:** extended likelihood graph $G = (V \cup F, D)$
- $F := \emptyset$
- **foreach** $v \in V$ **do**
  - $V_{next} := \{x \in V | \exists (v, x, l) \in D\}$
  - $D := D \setminus \{(v, y, l) \in D\}$
  - $E_r := \{e.r | \exists e \in t \in L$ with $e.a = v\}$
  - **foreach** $r \in E_r$ `// extend with resources (e.g., users)` **do**
    - $F := F \cup \{r\}$
    - $D := D \cup \{v, r,$`likeG`$(L, v, \varnothing, r, \varnothing,$"$resource$"$)\}$ `// likeG cf. Alg. 4`
    - $E_{wd} := \{getWkdy(e.t) | \exists e \in t \in L$ with $e.a = v \wedge x.r = r\}$
    - **foreach** $wd \in E_{wd}$ `// extend with weekdays` **do**
      - $F := F \cup \{wd\}$
      - $D := D \cup \{r, wd,$`likeG`$(l, v, \varnothing, r, \varnothing,$"$weekday$"$)\}$
      - **foreach** $v_{next} \in V_{next}$ `// connect to succeeding activities of` v **do**
        - $likly :=$`likeG`$(L, v, v_{next}, r, wd,$"$final$"$))$
        - **if** $likly > 0$ `// add edge only iff` v_{next} `follows on` r `and` wd **then**
          - $D := D \cup \{wd, v_{next}, likly\}$
- **return** $G = (V \cup F, D)$

Algorithm 3: Creating an extended likelihood graph based on a basic graph.

The proposed approach extends the basic likelihood graph, Fig. 3, to the extended likelihood graph depicted in Fig. 4 (using the expected execution events described in the running example, cf. Fig. 1). The extended graph enables to distinguish, for example, that user Eve and Tom have evenly shared their work on activity A (each one has executed the activity 2 times from 4 total executions).

Note, that the extended graph does not only represent the execution traces that are stored in the analyzed log. Instead it supports a more flexible activity focused representation so that, for example, traces that are theoretically possible and valid, but not defined in the execution logs, are also supported (e.g., that activity A is executed by Tom followed up by an execution of B by Adam).

Algorithm likeG(*a log* L, *activities* a_s, a_e, *resource* r, *weekday* wd, *likelihood type to calculate type* $\in \{resource, weekday, final\}$)

Result: likelihood that an event transition, based on a_s, a_e, r, wd, is observed in L

$tc := 0$, $ec := 0$

foreach $t \in L$ **do**

if $type =$ "$resource$" **then**

$tc{:=}tc + |\{x | x \in t.e \wedge x.a = a_s\}|$// count total executions of activity a_s

$ec{:=}ec + |\{x | x \in t.e \wedge x.a = a_s \wedge x.r = r\}|$// executions of a_s by resource r

else if $type =$ "$weekday$" **then**

$tc{:=}tc + |\{x | x \in t.e \wedge x.a = a_s \wedge x.r = r\}|$

$ec{:=}ec + |\{x | x \in t.e \wedge x.a = a_s \wedge x.r = r \wedge getWkdy(x.t) = wd\}|$

else if $type =$ "$final$" **then**

$tc{:=}tc + |\{x | x \in t.e \wedge x.a = a_s \wedge x.r = r \wedge getWkdy(x.t) = wd\}|$

$ec{:=}ec + |\{x | x \in t.e \wedge x.a = a_s \wedge x.r = r \wedge getWkdy(x.t) = wd \wedge x^+.a = a_e\}|$

return ec/tc

Algorithm 4: Likelihood calculating during extended graph construction.

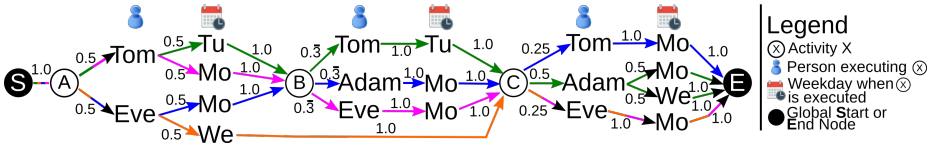

Fig. 4. Extended likelihood graph based on the running example.

We found, that, when compared with a "strict" representation that would only support the traces that were explicitly observed, the proposed approach enables to deal more flexibly with unexpected events, cf. Sect. 2.4.

2.3 Anomaly Detection in Ongoing Process Execution Events

Process model execution engines, such as the Cloud Process Execution Engine [11], generate fine granular[1] execution events that enable to monitor each ongoing execution in detail. These events, are not only stored in an execution log but are also available during an ongoing execution. Hence, immediately when, e.g., a process activity is selected for execution or a resource gets assigned to an activity, a related event becomes available and can be checked, by the proposed approach, for anomalies. Note, that events are identified as anomalies if they are *unlikely* compared to the expected execution events that are stored in the recorded process execution event log L and L's representation in G, cf. [2].

To determine if observed ongoing execution events are unlikely their likelihood is compared to the likelihood of comparable events in the recorded execution logs, based on the likelihood graph G. This is, *a*) an aggregated *ongoing likelihood* of all events that were generated by a process execution engine for the ongoing process model execution, cf. Alg. 5; and *b*) an aggregated *comparison likelihood* based on the recorded execution log traces in L, cf. Alg. 6, that show a comparable behavior to the ongoing execution (e.g., the traces cover at least the process activity that was most recently executed in the ongoing process model execution). Finally, both likelihoods are compared and iff the ongoing likelihood of the observed execution events is below the comparison likelihood then the ongoing execution events are identified as *unlikely* and, therefore, as anomalous.

[1] We expect that the ongoing execution events either represent an activity selection, resource assignment, or activity execution start timestamp.

The ongoing likelihood of each *ongoing process execution* event is calculated by Alg. 5. Alg. 5 utilizes the extended likelihood graph $G = (V, D)$, an execution logs L, a vertex $lst_v \in V$ which represents the last observed event in the ongoing process execution that was successfully mapped onto G, a vertex $lst_{va} \in V$ which, analogously to lst_v, represents the last successfully mapped activity, the just executed/observed event f which should be mapped (i.e., an activity, resource, or execution start timestamp that should be retrieved in G), and the ongoing likelihood for all the already mapped events (i.e., preceding events of the ongoing execution) $lst_l \in (0; 1]$. Initially, when starting the anomaly detection/process execution lst_v is set to S and lst_l to 1.

Alg. 5 maps each observed process model execution event f (e.g., that a resource was chosen to execute an activity) onto the successors of lst_{va}, based on G. If the observed event is also present in the successors then the likelihood for the observed event can be extracted from the respective edge in D. If it is not present (i.e., $fnd = false$) then an unexpected event has been observed (e.g., a manual adaption of the process models' execution). Hence, the likelihood of the last observed event cannot be extracted from G and is, therefore, artificially calculated. Note, this case is addressed in Sect. 2.4. Finally, f's determined likelihood $likly$ is multiplied with the aggregated likelihood of the preceding events lst_l to represent the ongoing total likelihood of all observed ongoing process execution events. The updated likelihood (i.e., $lst_l * likly$) is returned together with the updated lst_v and lst_{va}. Imagine that events were observed which indicate that activity A was executed by Tom on a Tuesday. Then the likelihood for these events can be calculated, based on G, as 1 ("S" $\rightarrow$ "A") times 0.5 ("A" $\rightarrow$ "Tom") times 0.5 ("Tom" $\rightarrow$ "Tu") equals 0.25, cf. Fig. 4.

The *comparison likelihood* l_c is calculated by Alg. 6 based on all recorded execution traces[2] $t \in L$ by mapping them on the likelihood graph G while aggregating t's respective event likelihoods in l_c. Finally, the minimum l_c likelihood of all traces is returned. Hence, the traces which were utilized to construct G are *never* identified as anomalous when applying Alg. 5 on them. Note, the likelihood aggregation is, for each trace, executed for exactly S_{max} events $e \in t$ where S_{max} is the exact number of events which were observed in the ongoing process model execution. Moreover, after S_{max} events the mapping must end up at the activity vertex $a \in V$ (i.e., lst_{va} when comparing if with an likelihood calculated by Alg. 5). Hence, the comparison likelihood can currently only be calculated after an activity selection event was observed. This limitation will be addressed in future work. The amount of events S_{max} is utilized to *a*) support traces t which contain loops were the same activity a is executed multiple times, i.e., the likelihood calculation should not stop if the searched activity is found the first time; and *b*) get a realistic comparison likelihood because the ongoing likelihood depends on the amount of aggregated events. For example, the minimum like-

[2] Note, this paper calculates the comparison likelihood based on the recorded traces in L because they represent expected execution event traces. Alternatively, for example, each theoretically possible trace (event order) in G could be constructed/analyzed.

```
Algorithm MapEvents(extended likelihood graph G = (V, D), a log L, vertexes
 lst_v, lst_va ∈ V that represent the last mapped event (lst_v) and the last activity
 selection event (lst_va), new event to map f, last ongoing execution likelihood lst_l)
    Result: updated lst_v, lst_va, and lst_l based on f
    D := {x ∈ D|x.v_s = lst_v}, fnd:=false, likly:=0
    foreach d ∈ D do
        if d.v_e = f // f is a successor of the last successfully mapped event lst_v then
            lst_v:=d.v_e, likly:=d.l, fnd:=true
            break
    if fnd=false // if the event f was not recorded in L then
        // calculate an artificial likelihood for f, see Sect. 2.4
        pun:=isActivity(f)?punAct:punOth // punishment factor selection
        if lst_va ≠ null // if the last activity event was retrieved in V then
            f_avgLkli := {x.l ∈EvntTypLkly(G, f, lst_va)|x.y = f}
            // EvntTypLkly cf. Alg. 8
            if |f_avgLkli| ≠ 0 // apply similarity likelihood then
                likly := f^0_avgLkli * pun
            else
                // apply Gini and class distribution likelihood
                likyhds := {x.l|x ∈EvntTypLkly(G, f, lst_va)}
                gLkli:=1−Gini(sort(likyhds), |likyhds|) // Gini cf. Eq. 1
                cLkly:=1−ClassLkly(L,f,lst_va,lst_v) // ClassLkly cf. Alg. 7
                likly := gLkli * cLkly * pun
        else
            // fallback if the last observed activity event was not found in G
            likly:=lst_l * pun
    if isActivity(f) // try to identify f in G iff its an activity then
        matchingActivities := {x ∈ V|x = f}
        if |matchingActivities| > 0 then
            lst_v:=matchingActivities^0; lst_va := lst_v
        else
            lst_va:=null
    return lst_v, lst_va, likly * lst_l
```

Algorithm 5: Mapping of execution event f on a likelihood graph G.

lihood l_c to reach vertex C in Fig. 4 based on the traces in Fig. 1 is found by mapping trace I1, I3, or I4 (all three traces have the same minimal likelihood).

Note that we propose that the comparison likelihood must be *substantially* below the likelihood of the ongoing execution events to identify an execution as anomalous. We assume that it is more beneficial to miss a single anomaly then to incorrectly mark 10 non-anomalous executions as anomalous (i.e., false positives). False positives could result in flooding, for example, fraud prevention departments with more cases then they can realistically investigate and, hereby, limit an organizations performance, cf. [9], or it can reduce the trust in the generated anomaly detection results. Nevertheless we are aware that the meaning of substantially can differentiate between multiple domains and data sources. Hence the presented approach introduces a user choose able factor $MinFac \in [0; 1]$. This factor is multiplied with the calculated comparison likelihood before comparing it with the ongoing likelihood of the ongoing execution events. Hereby it controls how unlikely the observed events can become before identifying them as an anomaly. Hence, a $MinFac$ of 0.0001 would result in a relaxed anomaly detection while a $MinFac$ value of 1 results in a strict analysis.

2.4 Dealing with Unexpected Execution Events

This paper found that process execution events can roughly be classified into events that are already represented by the generated likelihood graph; and events

Algorithm MinLike(*a log* L, *a likelihood graph* $G = (V, D)$, *activity to look for* a, *max amount of events to map* S_{max})
  Result: minimal likelihood $min \in [0; 1]$ to reach activity a
  min:=1
  foreach $t \in L$ **do**
    s_c:=0, $found_a$:=$false$, l_c:=1
    foreach $e \in t$ // map t on G to determine a comparison likelihood l_c **do**
      s_c:=$s_c + 1$
      // G's likelihood for e's resource, weekday, and the succeeding activity
      l_{AtoR}:=$\{x \in D | x.s_v = e.a \wedge x.e_v = e.r\}^0$
      l_{RtoWD}:=$\{x \in D | x.s_v = l_{AtoU}.e_v \wedge x.e_v = getWkdy(e.t)\}^0$
      l_{WDtoA}:=$\{x \in D | x.s_v = l_{RtoWD}.e_v \wedge x.e_v = e^+.a\}^0$
      // aggregate likelihoods for e based on G
      l_c:=$l_c * l_{AtoU}.l * l_{RtoWD}.l * l_{WDtoA}.l$
      if $e.a = a \wedge s_c = S_{max}$ **then**
        $found_a$:= $true$; break
      else if $s_c > S_{max}$ **then**
        break
    if $found_a = true \wedge l_c < min$ **then**
      min:=l_c;
  return min

Algorithm 6: Calculation of minimal path likelihood for a chosen activity.

that are not (i.e., unexpected events). Especially, for the second kind of events it can further be divided in events that are valid (e.g., in comparison to the process model, for example, parallel executions can lead to valid varying nondeterministic, and therefore unexpected, activity execution event orders) and events that are invalid (e.g., that activities execution events are observed in a wrong order).

We noticed that real life process execution data frequently contains valid but still unexpected events (i.e., an event that is novel, so it was not represented in the analyzed execution logs). This kind of events can but do not necessarily indicate a harmful anomaly. For example, assume that an activity was, until now, always executed on Monday. Suddenly it is executed on Sunday when all employees enjoy their free weekend, likely this is an harmful anomaly that stems from an attack (e.g., credentials could have been stolen through phishing) and should be reported. However, if the same activity is executed on a Thursday then this could stem from a large customer order and terminating the process because of a detected, but harmless, anomaly would negatively affect the organization.

Existing work frequently marks any kind of unexpected event as anomalous. However, as described, this is not always beneficial so that this work proposes approaches, utilized by Alg. 5, to calculate an artificial likelihood for unexpected events. This enables the anomaly detection algorithms to take unexpected events into account – in a diverse, flexible, and context dependent way. Alg. 5 starts iff an unexpected event is observed (i.e., $fnd = false$) the artificial likelihood calculation by selecting a *punishment factor*. Subsequently the calculated artificial likelihood is multiplied with the user chosen punishment factor, hereby, $punAct \in [0; 1]$ is utilized for activity execution events and $punOth \in [0; 1]$ for the other observed execution event process perspectives. This represents that the artificial likelihood was not determined based on the recorded real execution log files and can indicate an harmful deviation from the expected events. Which of the available artificial likelihood calculation approaches is applied depends on the available data, e.g., if the last executed activity lst_{va} was found in G.

The proposed approach integrates one standard approach (Gini) and three proposed approaches for artificial likelihood calculation: *a*) *Gini Coefficient* – measures the statistical dispersion of multiple likelihoods (e.g., if an activity is almost always executed on Monday then an execution on a novel weekday is assumed as more unlikely than if the executions were evenly spread over most possible weekdays) – $Gini(x, n)$ where x is an ordered list of likelihoods and n is x's size (cf. Eq. 1); and *b*) *Class Distribution* – we assume that if a very high or low amount of event class instances (e.g., weekdays) is already utilized then it is unlikely that a novel class instances will be observed. For example, assume that an activity is executed on all weekdays except Sunday. Then the proposed approach assesses that an sudden execution on Sunday is unlikely because we expect that it was left out for purpose in the recorded execution logs L, cf. Alg. 7; and *c*) *Similarity Likelihood* – Alg. 8 determines the likelihood for an unexpected event based on the expected events for a chosen activity. For example, assume that a user u has never executed activity a on Monday, while for other users such an execution date, for a, is frequently observable. Then it can be assumed that u's Monday execution is rather likely. In Alg. 8, `getType`(v) determines if v represents an activity, resource, or execution start time (i.e., a weekday), i.e., $getType : V \rightarrow \{activity, resource, weekday\}$; and *d*) *Likelihood Forecasting fall-back* – utilizes the previously calculated/known likelihood of the last processed event (e.g., an activity execution start weekday) to determine the likelihood of the currently analyzed event, cf. Alg. 5.

The first three approaches (Gini, Class, and Similarity) are applied if the last activity execution event a (i.e., an event generated by an ongoing process execution that is monitored for anomalies) can be retrieved in the recorded expected execution logs (i.e., an execution of a is stored "in" the logs L that are represented by the likelihood graph G). This is because these three approaches utilize recorded expected activity execution events to calculate the likelihood of unexpected events. Hence, if an unexpected activity is executed (e.g., a valid execution path is utilized that is not present in the analyzed logs) then the last forecasting based approach is applied as a fall-back. Note, all four approaches can be applied to calculate an artificial likelihood for all possible unexpected events (e.g., resources allocation events or activity execution time events).

Note, the proposed approach supports all discussed event types (e.g., expected and unexpected events) along with all three described anomaly types, i.e., *a*) *Point Anomalies* – by applying a punishment factor for unexpected events so that a deviation from the expected events can identify a process models' execution as anomalous; and *b*) *Contextual Anomalies* – because the proposed approach integrates multiple process perspectives to analyze each event with respect to its specific context; and *c*) *Collective Anomalies* – because the proposed approach aggregates the likelihoods of each observed execution event so that single small deviations will, while a process model is executed, aggregate until the observed events get so unlikely that they are identified as anomalous.

$$Gini(x, n) = \frac{n}{n-1} \cdot \frac{2 \cdot \sum_{i=1}^{n} i x_i}{\sum_{i=1}^{n} x_i} - \frac{n+1}{n} \tag{1}$$

Algorithm ClassLkly(*log L, an unexpected event f, $lst_v, lst_a \in V$ which represent the last event (lst_v) and activity event (lst_a) that were successfully mapped onto G*)

Result: artificial class likelihood of f for activity lst_a
tc:=0, ec:=0
foreach $t \in L$ **do**
  if *isRes(f)* // if f represents a resource assignment **then**
    tc:=$tc + |\{x.r | e \in t \wedge x \in e\}|$ // resource count for whole process
    ec:=$ec + |\{x.r | e \in t \wedge x \in e \wedge e.a = lst_a\}|$ // resource count for activity lst_a
  else if *isWeekday(f)* // if f represents an execution start timestamp **then**
    tc:=$tc + |\{getWkdy(x.t) | e \in t \wedge x \in e\}|$
    if *isRes(lst_v)* **then**
      ec:=$ec + |\{getWkdy(e.t) | e \in t \wedge x \in e \wedge e.a = lst_a \wedge e.r = lst_v\}|$
    else
      ec:=$ec + |\{getWkdy(e.t) | e \in t \wedge x \in e \wedge e.a = lst_a\}|$
  else
    // if f represents an activity selection
    tc:=$tc + |\{x^+.a | e \in t \wedge x \in e \wedge e.a = lst_a\}|$
    if *isRes(lst_v)* **then**
      ec:=$ec + |\{x^+.a | e \in t \wedge x \in e \wedge e.a = lst_a \wedge e.r = lst_v\}|$
    else
      ec:=$ec + |\{x^+.a | e \in t \wedge x \in e \wedge e.a = lst_a \wedge getWkdy(e.t) = lst_v\}|$
// calculating the artificial likelihood based on the tc and ec
max:=0.5, min:=$1/(tc+1)$, $rawLkly$:=ec/tc
if $rawLkly > max$ **then**
  $rawLkly$:=$1 - rawLkly$
return $((rawLkly) - min)/(max - min)$

Algorithm 7: Class distribution based artificial likelihood calculation.

Algorithm EvntTypLkly(*likelihood graph $G = (V, D)$, unexpected event f, activity v_{lstA}*)

Result: set of tuples (v, l) of vertexes $v \in V$, of f's type, and their likelihood l for v_{lstA}
E:=$\{(d, 1) | d \in D \wedge d.v_s = v_{lstA}\}$
$fnd_{fs} := \emptyset$, $kwn_E := \emptyset$
foreach $e \in E$ **do**
  $kwn_E := kwn_E \cup \{e.d\}$;
  if *getType($e.d.v_e$) = getType(f)* // look for f's type **then**
    $fnd_f := \{x \in fnd_{fs} | x.v_e = e.d.v_e\}$
    if $fnd_f = \emptyset$ **then**
      fnd_{fs}:=$fnd_{fs} \cup \{(e.d.v_e, e.l)\}$
    else
      fnd_{fs}:=$fnd_{fs} \setminus fnd_f \cup \{(e.d.v_e, fnd_f^0.l * e.d.l)\}$
  else
    $E := \{(x, e.d.l * x.l) | x \in D \wedge x.v_s = e.d.v_e \wedge x \notin kwn_E\}$
return fnd_{fs}

Algorithm 8: Artificial likelihood that f is observed for activity v_{lstA}.

When applying the presented approach on the running example then the scenario depicted in Fig. 5 can be observed. Hence, some of Eve's anomalous behavior (the bottom red path) can be found in the generated extended likelihood graph while others can't (i.e., unexpected events, bottom red dotted path). So, for the unexpected events an artificial likelihood is calculated. For the sake of brevity the artificial likelihoods are only calculated for three of the five occurrences of unexpected events, i.e., ①, ③, and ⑤. A combination of Gini and Class Distribution is applied on ①. The Similarity Likelihood approach is illustrated based on ③. Finally, the artificial likelihood of ⑤ is calculated by means of Likelihood Forecasting. Note, that *punOth* is assumed as 0.5, for this example.

The Gini Coefficient, which is 0 (i.e., the likelihoods are evenly spread over the possible paths), for ① is calculated by applying Eq. 1 on the weekday likelihoods (0.5 for "Mo" and 0.5 for "We") for "Eve" and activity "A". The Class Distribution is calculated as $0.\overline{3}$ based on an $ec = 2$ (weekdays used by "Eve" for activity "A") and $tc = 3$ (total weekdays used for activity A). Overall the

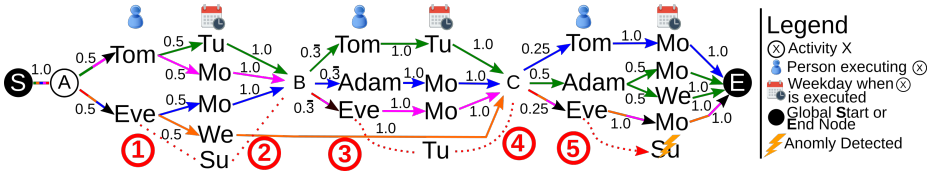

Fig. 5. Anomaly detection using the extended graph, based on the running example.

artificial likelihood is calculated as Gini $(1-0)$ times Class Distribution $(1-0.\overline{3})$ times the punishment factor (0.5) as $0.\overline{3}$, cf. Alg. 5.

③ is utilized to illustrate the Similarity Likelihood based approach. Note, that "Eve" executed the activity "B" on a Tuesday ("Tu") which she never did before, based on L. Hence, the similarity based approach analyses how likely "B" is overall executed on a Tuesday – which is $0.\overline{3}$ (1 out of 3 observed executions). So, the similarity likelihood is $0.\overline{3}$ times the punishment factor $(0.5) \mapsto 0.1\overline{6}$.

Finally, when Likelihood Forecasting is applied on ⑤ the last determined aggregated ongoing execution likelihood lst_l is multiplied with the punishment factor so that $punOth = 0.5$ can be utilized as the artificial likelihood for this single event (i.e., transition likelihood from "Eve" to "Su").

3 Evaluation

The evaluation utilizes real life process execution data (i.e., execution logs) to assess the anomaly detection performance/applicability of the proposed approach.

Test Problems The evaluation utilizes real life process execution logs from the BPI Challenge 2015[3] (BPIC) and Higher Education Processes (HEP), cf. [10].

The BPIC logs consist of 262,628 events which belong to 5,649 execution traces and 398 activities – recorded from 2010 to 2015. The logs were provided by five (BPIC_1 to BPIC_5) Dutch building authorities. The HEP logs contain 28,129 events, 354 execution traces, and 147 activities – recorded from 2008 to 2011. Each logged trace records the interactions of one student during a particular course (e.g., forum posts or exercise uploads) with a learning platform. Each recorded year is stored in an independent execution log file (HEP_1 to HEP_3). Note, all the BPIC and HEP logs contain the information required by the proposed approach (i.e., resources, timestamps, and the control flow).

Each of the logs was evenly and randomly separated into training (utilized to generate the likelihood graph) and test data (used to evaluate the performance of the proposed approach, i.e., if anomalous and non-anomalous traces are correctly identified). Moreover, fifty percent of the test data execution traces were mutated to construct execution traces with anomalous execution events. For this four anomaly mutators are proposed that focus on the control flow, the organizational perspective, and the execution timestamps: *a*) *Altered Activity Order* – an activity is moved to a new position in the mutated test data execution trace; and *b*) *New Activity* – a new activity is added into the mutated execution trace; and

[3] `http://www.win.tue.nl/bpi/2015/challenge`—DOI: 10.4121/uuid:31a308ef-c844-48da-948c-305d167a0ec1

c) *New Date* – the execution start timestamp of an recorded execution event is changed; and *d*) *New Resource* – a new, novel, resource is assigned to an activity execution event. Mutations were conducted on test data execution traces based on a randomly chosen anomaly density: *a*) *Low* – only a single anomaly mutator is applied two, three or four times; and *b*) *Medium* – two of four anomaly mutators are applied one, two or three times; and *c*) *High* – three of four anomaly mutators are applied one or two times.

We opted to combine multiple anomaly mutators and densities to represent that in real life data each anomaly is unique and affects different parts and perspectives of a process model execution with different strength. This enables to evaluate the detection of point, collective, and contextual anomalies. Moreover, the correct handling of unexpected events is analyzed because the test data also contains valid events that are not represented in the training data and, therefore, in the likelihood graph (e.g., manual ad-hoc process changes that should not be identified as an anomaly). Note, the anomalies were randomly generated. Hence, the evaluation was executed 100 times for each log to even out random behavior – the average of all evaluation runs is used in the following.

Metrics and Evaluation The evaluation assesses the anomaly detection capabilities of the presented approach. Hence, a likelihood graph is generated from the training execution traces and, subsequently, applied to distinguish non-anomalous from anomalous test traces. Four key values are determined: True Positive (TP, how many anomalous execution traces were correctly identified), False Positive (FP, how many non-anomalous traces were incorrectly identified as anomalous), True Negative (TN, how many non-anomalous traces were correctly identified), False Negative (FN, how many anomalous traces were incorrectly identified as non-anomalous) for each log file (i.e., BPIC_1-5 and HEP_1-3).

Based on the presented key values three standard metrics, cf. [8], are calculated: *a*) *Precision (P)* $= TP/(TP + FP)$ – indicates if the identified anomalies were also real anomalies; and *b*) *Recall (R)* $= TP/(TP + FN)$ – indicates if anomalies were "overlooked" (i.e., if anomalous traces were not identified as such); and *c*) *Accuracy (A)* $= (TP + TN)/\ (TP + TN + FP + FN)$ – provides a general anomaly detection performance overview; $P, R, A \in [0; 1]$.

For this paper we assume that the number of False Positives should be as low as possible so that the "precision" becomes close to 1. In addition the F_β-measure, Eq. 2, metric is applied because it provides a configurable harmonic mean between Precision and Recall, cf. [8]. Hereby, β controls the balance between P and R. If $\beta = 1$ then a harmonic mean between P and R is calculated. If $\beta < 1$ then F_β becomes precision-oriented and if $\beta > 1$ then it becomes recall oriented. A $F_{0.5}$-measure and a F_1-measure were used during the evaluation.

$$F_\beta = \frac{(\beta^2 + 1) \cdot P \cdot R}{\beta^2 \cdot P + R} \tag{2}$$

Results The results were generated by analyzing the BPIC and HEP execution log files with a proof-of-concept implementation of the presented approach. Overall, the algorithms needed less than 5 minutes to calculate the following results

on a standard 2.6 Ghz Intel Q6300 CPU with 8 GB of RAM. This suggests an applicability on larger process repositories/execution logs. Note, the likelihood graph must be, for example, only generated once a year – based on the most recent recorded process execution logs and, hereby, based on the most recent and, therefore, most relevant events. Subsequently it can be used to identify anomalies in multiple succeeding process model executions.

Primary tests were executed to identify appropriate configuration values for the presented approach. The punishment factor for activities was set to $punAct = 0.9$. For non-activity data it was set to $punOth = 0.95$. Note, a lower punishment factor results in a stronger punishment. Hence, a *punAct* was configured more strict then *punOth* to represent that an unexpected activity has a higher, assumed, impact on a process execution then an unexpected user or execution weekday. Finally, the minimal anomaly factor was $MinFac = 1 \times 10^{-5}$ so that the calculated likelihood for the ongoing execution events must be substantially below the determined expected comparison likelihood (i.e., to reduce the amount of false positives at a price of a slightly higher false negative rate). A more relaxed punishment factor or/and a higher minimal anomaly factor would increase the recall metric but lower the precision metric score.

The average results of the evaluation are shown in Tab. 1. The precision metric reached an average result of 82%. Also the other metrics show promising results (on average 65% for recall and 78% for accuracy) so that it can be concluded that the proposed approach can successfully be applied on complex real world process logs. Moreover, we analyzed the runtime anomaly detection capacities of the presented approach. It was found that the anomalous test data execution traces were identified, as anomalous, substantially before all trace events were analyzed (i.e., before the anomalous process executions finished).

Table 1. Anomaly detection performance of the presented approach.

	BPIC_1	BPIC_2	BPIC_3	BPIC_4	BPIC_5	HEP_1	HEP_2	HEP_3
Precision	0.79	0.92	0.81	0.78	0.83	0.80	0.83	0.81
Recall	0.62	0.71	0.57	0.72	0.63	0.75	0.56	0.60
Accuracy	0.77	0.73	0.77	0.79	0.79	0.82	0.80	0.77
F_1-measure	0.69	0.8	0.67	0.75	0.72	0.77	0.67	0.68
$F_{0.5}$-measure	0.75	0.87	0.74	0.77	0.78	0.79	0.76	0.76

4 Related Work

Existing process anomaly detection work can be divided in *a*) approaches that focus on the process control flow; and *b*) approaches which cover also more diverse anomaly scenarios/perspectives. Papers, in *a*), typically strive to ensure an anomaly-free control flow activity execution order (cf. [4, 2, 1, 16, 3]). Therefore, they generate a model that represents an expected control flow activity order,

for example, through applying standardized process mining or custom statics based approaches. Hence, they ignore a wide range of non-control flow related data (e.g., the time or organizational perspective) which could, e.g., result in undetected fraud attempts. Moreover, they mark a process execution as anomalous as soon as any unexpected execution event is observed.

The second category, *b*), contains approaches that integrate non-control flow related data, such as, external sensors (cf. [19, 12, 15, 14, 13]). Therefore, existing work typically applies a diverse set of strategies that are strongly focused on one specific use case/domain – which reduces their general applicability. For example, [12] analyzed payment transactions to identify money laundering by checking for conspicuous transaction orders (i.e., a large quantity of very small transactions that all end up at the same account). Other approaches focus only on temporal anomalies (e.g., surprisingly short activity execution times, cf. [13]). Moreover, the identified existing work is limited to single aspects, such as only temporal or sensor data, hence a combined representation and analysis of multiple process perspectives at once, as provided by the proposed approach, is not supported.

Finally, existing work frequently focuses on point anomalies. Consequently, attacks which carefully and gradually compromise ongoing process executions (i.e., collective anomalies) will unlikely be detected. Moreover, existing work has difficulties when dealing with unexpected events. Typically, this results in a huge amount of detected, probably harmless, "anomalies" (e.g., more then 500,000 "anomalies" were identified by [12] in a log that only covers a short timespan) so that the responsible security/fraud departments are overwhelmed with a, potentially, unmanageable workload. In comparison, the presented approach enables a fine granular and automatic handling of unexpected events which leads to an anomaly detection precision of about 82%, cf. Sect. 3. Finally, the presented approach provides a tight integration of the control-flow perspective along with execution timestamps and resources – which is not achieved by existing work.

5 Conclusion

The presented process execution event anomaly detection approach ($\mapsto$ **RQ1** to **RQ3**) represents expected execution events and its occurrence likelihood (based on recorded execution log files) in a likelihood graph. In the following, succeeding process model execution events are mapped onto the likelihood graph to identify if the observed execution events are unlikely and could, therefore, indicate an anomaly. This enables to tackle novel areas in the process anomaly detection domain, for example, runtime anomaly detection, how contextual and collection anomalies can be detected ($\mapsto$ **RQ2**), and ways to deal with unexpected events ($\mapsto$ **RQ3**). The evaluation supports the applicability and feasibility of the presented approach for real life process data from multiple domains.

Future work will strive to enhance the computational performance of the presented approach, because currently each likelihood graph must be regenerated from scratch whenever novel process execution traces should be integrated. So we plan to propose update techniques that can integrate novel, non-anomalous,

traces in an existing likelihood graph. Moreover we plan to evaluate the applicability of the presented approach in inter-organizational large scale scenarios.

References

1. Bezerra, F., Wainer, J.: Anomaly detection algorithms in business process logs. In: Enterprise Information Systems. pp. 11–18 (2008)
2. Bezerra, F., Wainer, J.: Anomaly detection algorithms in logs of process aware systems. In: Applied computing. pp. 951–952. ACM (2008)
3. Bezerra, F., Wainer, J.: Algorithms for anomaly detection of traces in logs of process aware information systems. Information Systems 38(1), 33–44 (2013)
4. Bezerra, F., Wainer, J., van der Aalst, W.M.: Anomaly detection using process mining. In: Enterprise, Business-Process and Information Systems Modeling, pp. 149–161. Springer (2009)
5. Böhmer, K., Rinderle-Ma, S.: Automatic signature generation for anomaly detection in business process instance data. In: Enterprise, Business-Process and Information Systems Modeling, pp. 184–199. Springer (2016)
6. Chandola, V., Banerjee, A., Kumar, V.: Anomaly detection: A survey. ACM Computing Surveys 41(3), 15 (2009)
7. Chatfied, C., Collins, A.J.: Introduction to multivariate analysis. Springer (2013)
8. Chinchor, N., Sundheim, B.: Muc-5 evaluation metrics. In: Message understanding. pp. 69–78. Association for Computational Linguistics (1993)
9. Jans, M., van der Werf, J.M., Lybaert, N., Vanhoof, K.: A business process mining application for internal transaction fraud mitigation. Expert Systems with Applications 38(10), 13351–13359 (2011)
10. Ly, L.T., Indiono, C., Mangler, J., Rinderle-Ma, S.: Data transformation and semantic log purging for process mining. In: Advanced Information Systems Engineering. pp. 238–253. Springer (2012)
11. Mangler, J., Rinderle-Ma, S.: Cpee-cloud process exection engine (2014)
12. Rieke, R., Zhdanova, M., Repp, J., Giot, R., Gaber, C.: Fraud detection in mobile payments utilizing process behavior analysis. In: Availability, Reliability and Security. pp. 662–669. IEEE (2013)
13. Rogge-Solti, A., Kasneci, G.: Temporal anomaly detection in business processes. In: Business Process Management, pp. 234–249. Springer (2014)
14. Sarno, R., Sinaga, F.P.: Business process anomaly detection using ontology-based process modelling and multi-level class association rule learning. In: Computer, Control, Informatics and its Applications. pp. 12–17. IEEE (2015)
15. Sinaga, F., Sarno, R.: Business process anomali detection using multi-level class association rule learning. Technology and Science 2(1) (2016)
16. Sureka, A.: Kernel based sequential data anomaly detection in business process event logs. preprint arXiv:1507.01168 (2015)
17. Van Der Aalst, W.: Process mining: discovery, conformance and enhancement of business processes. Springer Science & Business Media (2011)
18. Weijters, A.J., Van der Aalst, W.M.: Rediscovering workflow models from event-based data using little thumb. Integrated Computer-Aided Engineering 10(2), 151–162 (2003)
19. Yang, W.S., Hwang, S.Y.: A process-mining framework for the detection of healthcare fraud and abuse. Expert Systems with Applications 31(1), 56–68 (2006)
20. Yu, L., Liu, H.: Efficient feature selection via analysis of relevance and redundancy. Machine Learning Research 5, 1205–1224 (2004)